JavaScript 07 – Objetos nativos

Introducción

En este tema vamos a ver:

- Las funciones globales de Javascript, muchas de las cuales ya hemos visto como *Number()* o *String*().
- Los objetos nativos que incorpora Javascript y que nos facilitarán el trabajo proporcionándonos métodos y propiedades útiles para no tener que "reinventar la rueda" en nuestras aplicaciones.
- ro de ellos está el objeto **RegExpr** que nos permite trabajar con **expresiones regulares** (son iguales que en otros lenguajes) que nos serán de gran ayuda, sobre todo a la hora de validar formularios y que por eso veremos en la siguiente unidad.

Funciones globales

- `parseInt(valor)`: devuelve el valor pasado como parámetro convertido a entero o *NaN* si no es posible la conversión. Este método es mucho más permisivo que *Number* y convierte cualquier cosa que comience por un número (si encuentra un carácter no numérico detiene la conversión y devuelve lo convertido hasta el momento). Ejemplos:

```
console.log( parseInt(3.84) )           // imprime 3 (ignora los decimales)
console.log( parseInt('3.84') )         // imprime 3
console.log( parseInt('28manzanas') )   // imprime 28
console.log( parseInt('manzanas28') )   // imprime NaN
```

- `parseFloat(valor)`: igual, pero devuelve un número decimal. Ejemplos:

```
console.log( parseFloat(3.84) )           // imprime 3.84
console.log( parseFloat('3.84') )         // imprime 3.84
console.log( parseFloat('3,84') )         // imprime 3 (la coma no es un carácter numérico)
console.log( parseFloat('28manzanas') )   // imprime 28
console.log( parseFloat('manzanas28') )   // imprime NaN
```

- `Number(valor)`: convierte el valor a un número. Es como *parseFloat* pero más estricto. Si no puede convertir todo el valor, devuelve *NaN*. Ejemplos:

```
console.log( Number(3.84) )            // imprime 3.84
console.log( Number('3.84') )          // imprime 3.84
console.log( Number('3,84') )          // imprime NaN (la coma no es un carácter numérico)
console.log( Number('28manzanas') )    // imprime NaN
console.log( Number('manzanas28') )    // imprime NaN
```

- `String(valor)`: convierte el valor pasado en una cadena de texto. Si le pasamos un objeto llama al método *.toString()* del objeto. Ejemplos:

```
console.log( String(3.84) )                // imprime '3.84'
console.log( String([24, 3. 12]) )         // imprime '24,3,12'
console.log( {nombre: 'Marta', edad: 26} )  // imprime "[object Object]"
```

- `Boolean(valor)`: convierte el valor pasado a un booleano. Sería el resultado de tenerlo como condición en un `if`. Muchas veces en vez de usar esto usamos la doble negación `!!` que da el mismo resultado. Ejemplos:

```
console.log( Boolean('Hola') )     // Equivaldría a !!'Hola'. Imprime true
console.log( Boolean(0) )          // Equivaldría a !!0. Imprime false
```

- `isNaN(valor)`: devuelve *true* si lo pasado NO es un número o no puede convertirse en un número. Ejemplos:

```
console.log( isNaN(3.84) )           // imprime false
console.log( isNaN('3.84') )         // imprime false
console.log( isNaN('3,84') )         // imprime true (la coma no es un carácter numérico)
console.log( isNaN('28manzanas') )   // imprime true
console.log( isNaN('manzanas28') )   // imprime true
```

- `isFinite(valor)`: devuelve *false* si es número pasado es infinito (o demasiado grande)

```
console.log( isFinite(3.84) )          // imprime true
console.log( isFinite(3.84 / 0) )          // imprime false
```

- `encodeURI(string)` / `decodeURI(string)`: transforma la cadena pasada a una URL codificada válida, transformando los caracteres especiales que contenga, excepto , / ? : @ & = + $ #. Debemos usarla siempre que vayamos a pasar una URL. Ejemplo:
  - Decoded: “http://domain.com?val=1 2 3&val2=r+y%6”
  - Encoded: “http://domain.com?val=1%202%203&val2=r+y%256”
- `encodeURIComponent(string)` / `decodeURIComponent(string)`: transforma también los caracteres que no transforma la anterior. Debemos usarla para codificar parámetros, pero no una URL entera. Ejemplo:
  - Decoded: “http://domain.com?val=1 2 3&val2=r+y%6”
  - Encoded: “http%3A%2F%2Fdomain.com%3Fval%3D1%202%203%26val2%3Dr%2By%256”

Objetos nativos del lenguaje

En Javascript casi todo son objetos. Ya hemos visto diferentes objetos:

- window

- screen
- navigator
- location
- history
- document

Los 5 primeros se corresponden al modelo de objetos del navegador y *document* se corresponde al modelo de objetos del documento. Todos nos permiten interactuar con el navegador para realizar distintas acciones.

También tenemos los tipos de objetos nativos, que no dependen del navegador. Son:

- Number
- String
- Boolean
- Array
- Function
- Object
- Math
- Date
- RegExp

Además de los tipos primitivos de **número, cadena, booleano, undefined** y **null**, Javascript tiene todos los objetos indicados. Como ya hemos visto, se puede crear un número usando su tipo primitivo (`let num = 5`) o su objeto (`let num = new Number(5)`) pero es mucho más eficiente usar los tipos primitivos. Pero aunque lo creemos usando el tipo de dato primitivo se considera un objeto y tenemos acceso a todas sus propiedades y métodos. Ejemplo: `num.toFixed(2)`

Ya hemos visto las principales propiedades y métodos de *Number*, *String*, Boolean y *Array* y aquí vamos a ver los de **Math** y **Date** y en el apartado de validar formularios, las de **RegExp**.

Objeto Math

Proporciona constantes y métodos para trabajar con valores numéricos:

- constantes: `.PI` (número pi), `.E` (número de Euler), `.LN2` (algoritmo natural en base 2), `.LN10` (logaritmo natural en base 10), `.LOG2E` (logaritmo de E en base 2), `.LOG10E` (logaritmo de E en base 10), `.SQRT2` (raíz cuadrada de 2), `.SQRT1_2` (raíz cuadrada de 1⁄2). Ejemplos:

```
console.log( Math.PI )          // imprime 3.141592653589793
console.log( Math.SQRT2 )       // imprime 1.4142135623730951
```

- `Math.round(x)`: redondea x al entero más cercano
- `Math.floor(x)`: redondea x hacia abajo (5.99 → 5. Quita la parte decimal)
- `Math.ceil(x)`: redondea x hacia arriba (5.01 → 6)
- `Math.min(x1,x2,...)`: devuelve el número más bajo de los argumentos que se le pasan.
- `Math.max(x1,x2,...)`: devuelve el número más alto de los argumentos que se le pasan.
- `Math.pow(x, y)`: devuelve x^y (x elevado a y).
- `Math.abs(x)`: devuelve el valor absoluto de x.
- `Math.random()`: devuelve un número decimal aleatorio entre 0 (incluido) y 1 (no incluido).

  Si queremos un número entre otros rangos haremos: `Math.random() * (max - min) + min`
  O si lo queremos sin decimales, haremos: `Math.round(Math.random() * (max - min) + min)`

- `Math.cos(x)`: devuelve el coseno de x (en radianes).
- `Math.sin(x)`: devuelve el seno de x.
- `Math.tan(x)`: devuelve la tangente de x.
- `Math.sqrt(x)`: devuelve la raíz cuadrada de x

Ejemplos:

```
console.log( Math.round(3.14) )     // imprime 3
console.log( Math.round(3.84) )     // imprime 4
console.log( Math.floor(3.84) )     // imprime 3
console.log( Math.ceil(3.14) )      // imprime 4
console.log( Math.sqrt(2) )         // imprime 1.4142135623730951
```

Objeto Date

Es la clase que usaremos siempre que vayamos a trabajar con fechas. Al crear una instancia de la clase le pasamos la fecha que queremos crear o lo dejamos en blanco para que nos cree la fecha actual. Si le pasamos la fecha podemos pasarle:

- milisegundos, desde la fecha EPOCH
- cadena de fecha
- valor para año, mes (entre 0 y 11), día, hora, minutos, segundos, milisegundos

Ejemplos:

```
let date1=new Date()    // Mon Jul 30 2018 12:44:07 GMT+0200 (CEST) (es cuando he ejecutado la instrucción)
let date7=new Date(1532908800000)    // Mon Jul 30 2018 00:00:00 GMT+0200 (CEST) (miliseg. desde 1/1/1070)
let date2=new Date('2018-07-30')    // Mon Jul 30 2018 02:00:00 GMT+0200 (CEST) (la fecha pasada a las 0h. GMT)
let date3=new Date('2018-07-30 05:30')  // Mon Jul 30 2018 05:30:00 GMT+0200 (CEST) (la fecha pasada a las 05:30h.
local)
let date6=new Date('07-30-2018')    // Mon Jul 30 2018 00:00:00 GMT+0200 (CEST) (OJO: formato MM-DD-AAAA)
let date7=new Date('30-Jul-2018')    // Mon Jul 30 2018 00:00:00 GMT+0200 (CEST) (tb. podemos poner 'Julio')
let date4=new Date(2018,7,30)    // Thu Ago 30 2018 00:00:00 GMT+0200 (CEST) (OJO: 0->Ene,1->Feb... y a las 0h.
local)
let date5=new Date(2018,7,30,5,30)    // Thu Ago 30 2018 05:30:00 GMT+0200 (CEST) (OJO: 0->Ene,1->Feb,...)
```

EJERCICIO: Crea en la consola dos variables fecNac1 y fecNac2 que contengan tu fecha de nacimiento. La primera la creas pasando una cadena y la segunda pasando año, mes y día.

Cuando ponemos la fecha como texto, como separador de las fechas podemos usar *-, /* o *espacio*.

Como separador de las horas debemos usar *:*

Cuando ponemos la fecha cono parámetros numéricos (separados por ,) podemos poner valores fuera de rango que se sumarán al valor anterior. Por ejemplo:

```
let date=new Date(2018,7,41)    // Mon Sep 10 2018 00:00:00 GMT+0200 (CEST) -> 41=31Ago+10
let date=new Date(2018,7,0)     // Tue Jul 31 2018 00:00:00 GMT+0200 (CEST) -> 0=0Ago=31Jul (el
anterior)
let date=new Date(2018,7,-1)    // Mon Jul 30 2018 00:00:00 GMT+0200 (CEST) -> -1=0Ago-1=31Jul-1=30Jul
```

OJO con el rango de los meses que empieza en 0->Ene, 1->Feb,…,11->Dic

Tenemos métodos ***getter*** y ***setter*** para obtener o cambiar los valores de una fecha:

- **fullYear**: permite ver (*get*) y cambiar (*set*) el año de la fecha:

```
let fecha=new Date('2018-07-30')    // Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)
console.log( fecha.getFullYear() )  // imprime 2018
fecha.setFullYear(2019)             // Tue Jul 30 2019 02:00:00 GMT+0200 (CEST)
```

- **month**: devuelve/cambia el número de mes, pero recuerda que 0->Ene,…,11->Dic

```
let fecha=new Date('2018-07-30')    // Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)
console.log( fecha.getMonth() )     // imprime 6
fecha.setMonth(8)                   // Mon Sep 30 2019 02:00:00 GMT+0200 (CEST)
```

- **date**: devuelve/cambia el número de día:

```
let fecha=new Date('2018-07-30')    // Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)
console.log( fecha.getDate() )      // imprime 30
fecha.setDate(-2)                   // Thu Jun 28 2018 02:00:00 GMT+0200 (CEST)
```

- **day**: devuelve el número de día de la semana (0->Dom, 1->Lun, …, 6->Sáb). Este método NO tiene *setter*:

```
let fecha=new Date('2018-07-30')    // Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)
console.log( fecha.getDay() )       // imprime 1
```

- **hours, minutes, seconds, milliseconds**, : devuelve/cambia el número de la hora, minuto, segundo o milisegundo, respectivamente.
- **time**: devuelve/cambia el número de milisegundos desde Epoch (1/1/1970 00:00:00 GMT):

```
let fecha=new Date('2018-07-30')    // Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)
console.log( fecha.getTime() )      // imprime 1532908800000
fecha.setTime(1000*60*60*24*25)     // Fri Jan 02 1970 01:00:00 GMT+0100 (CET) (le hemos
añadido 25 días a Epoch)
```

EJERCICIO: Realiza en la consola los siguientes ejercicios (usa las variables que creaste antes)

- muestra el día de la semana en que naciste
- modifica fecNac1 para que contenga la fecha de tu cumpleaños de este año (cambia sólo el año)
- muestra el día de la semana de tu cumpleaños de este año
- calcula el nº de días que han pasado desde que naciste hasta hoy

Para mostrar la fecha tenemos varios métodos diferentes:

- **.toString()**: “Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)”
- **.toUTCString()**: “Mon, 30 Jul 2018 00:00:00 GMT”
- **.toDateString()**: “Mon, 30 Jul 2018”
- **.toTimeString()**: “02:00:00 GMT+0200 (hora de verano de Europa central)”
- **.toISOString()**: “2018-07-30T00:00:00.000Z”
- **.toLocaleString()**: “30/7/2018 2:00:00”
- **.toLocaleDateString()**: “30/7/2018”
- **.toLocaleTimeString()**: “2:00:00”

EJERCICIO: muestra en distintos formatos la fecha y la hora de hoy

NOTA: recuerda que las fechas son objetos y que se copian y se pasan como parámetro por referencia:

```
let fecha=new Date('2018-07-30')    // Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)
let otraFecha=fecha
otraFecha.setDate(28)               // Thu Jun 28 2018 02:00:00 GMT+0200 (CEST)
console.log( fecha.getDate() )      // imprime 28 porque fecha y otraFecha son el mismo objeto
```

Una forma sencilla de copiar una fecha es crear una nueva pasándole la que queremos copiar:

```
let fecha=new Date('2018-07-30')    // Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)
let otraFecha=new Date(fecha)
otraFecha.setDate(28)               // Thu Jun 28 2018 02:00:00 GMT+0200 (CEST)
console.log( fecha.getDate() )      // imprime 30
```

En realidad lo que le estamos pasando es el tiempo Epoch de la fecha (es como hacer `otraFecha=new Date(fecha.getTime())`)

NOTA: la comparación entre fechas funciona correctamente con los operadores >, >=, < y <=

Pero NO funciona con ==, ===, != y !== ya que compara los objetos y ve que son objetos diferentes.

Si queremos saber si 2 fechas son iguales (siendo diferentes objetos):

- el código que pondremos NO es `fecha1 === fecha2`
- pondremos `fecha1.getTime() === fecha2.getTime()`.

EJERCICIO: comprueba si es mayor tu fecha de nacimiento o el 1 de enero de este año.

Podemos probar los distintos métodos de las fechas en la página de w3schools.